Texas Instruments

TMS320C8x Software Tools

Release 1.13

Getting Started

1995

Microprocessor Development Systems

TMS320C8x Software Tools Getting Started

Release 1.13

TEXAS INSTRUMENTS

IMPORTANT NOTICE

Texas Instruments (TI) reserves the right to make changes to its products or to discontinue any semiconductor product or service without notice, and advises its customers to obtain the latest version of relevant information to verify, before placing orders, that the information being relied on is current.

TI warrants performance of its semiconductor products and related software to the specifications applicable at the time of sale in accordance with TI's standard warranty. Testing and other quality control techniques are utilized to the extent TI deems necessary to support this warranty. Specific testing of all parameters of each device is not necessarily performed, except those mandated by government requirements.

Certain applications using semiconductor products may involve potential risks of death, personal injury, or severe property or environmental damage ("Critical Applications").

TI SEMICONDUCTOR PRODUCTS ARE NOT DESIGNED, INTENDED, AUTHORIZED, OR WARRANTED TO BE SUITABLE FOR USE IN LIFE-SUPPORT APPLICATIONS, DEVICES OR SYSTEMS OR OTHER CRITICAL APPLICATIONS.

Inclusion of TI products in such applications is understood to be fully at the risk of the customer. Use of TI products in such applications requires the written approval of an appropriate TI officer. Questions concerning potential risk applications should be directed to TI through a local SC sales office.

In order to minimize risks associated with the customer's applications, adequate design and operating safeguards should be provided by the customer to minimize inherent or procedural hazards.

TI assumes no liability for applications assistance, customer product design, software performance, or infringement of patents or services described herein. Nor does TI warrant or represent that any license, either express or implied, is granted under any patent right, copyright, mask work right, or other intellectual property right of TI covering or relating to any combination, machine, or process in which such semiconductor products or services might be or are used.

Printed in U.S.A. by
Champion Press
Houston, Texas

TRADEMARKS

MS-DOS is a registered trademark of Microsoft Corporation.

Pentium is a trademark of Intel Corporation.

OpenWindows, Solaris, and SunOS are trademarks of Sun Microsystems, Inc.

SPARCstation is licensed exclusively to Sun Microsystems, Inc.

Windows NT is a trademark of Microsoft Corporation.

UNIX is a registered trademark of Unix System Laboratories, Inc.

Contents

Chapter 1

Installing the TMS320C8x Software Tools With SunOS

This chapter tells you how to install release 1.13 of the TMS320C8x software tools on a SPARCstation running OpenWindows™ under SunOS™ version 4.1.x (or higher). If you are using the Windows NT™ version of the software tools for the PC, turn to Chapter 2 for installation instructions.

The SunOS version of the TMS320C8x software tools package is composed of the following:

- ❑ The master processor (MP) and parallel processor (PP) C compilers
- ❑ The MP and PP assemblers
- ❑ The linker
- ❑ The MP and PP simulators
- ❑ The MP and PP C source debuggers
- ❑ The runtime-support and C I/O libraries

When you complete this installation, turn to Chapter 3 to verify the installation.

Chapter 4 contains documentation of tools and features that are new or have been changed since the last release. For more information about how to use the 'C8x tools, refer to the *TMS320C80 (MVP) Online Reference.*

Topic **Page**

1.1 What You'll Need

The following checklists describe items that are shipped with your 'C8x software tools and any additional items you'll need to use these tools.

Hardware checklist

□ **host** A SPARCstation or compatible system with SPARCstation 2 class or higher performance

□ **memory** 32 Mbytes of RAM

□ **disk space** 30 Mbytes of disk space for the software tools. In addition, you may need 1 Gbyte or more for software development or for working with digital images.

□ **display** Color monitor

□ **required hardware** Mouse

□ Keyboard

□ CD-ROM drive

□ **root privileges** You must have root privileges to mount and unmount the CD-ROM if you have SunOS 4.1.x, SunOS 5.0, or SunOS 5.1.

Software checklist

□ **operating system** OpenWindows version 3.0 (or higher) running under SunOS version 4.1.x (or higher). If you're using SunOS 5.x (also know as Solaris 2.x), you must have the Binary Compatibility Package (BCP) installed; if you don't, get your system administrator's help.

□ **interprocess communication features** Interprocess communication (IPC) features are included with your operating system. To verify that you have the IPC features enabled, enter **ipcs** from the command line; if you have the IPC features installed, you'll see a series of messages about shared memory and semaphores.

□ **make utility** If you have SunOS 5.x, you must install the UNIX make utility.

□ **CD-ROMs** *TMS320C80 (MVP) Online Reference*

□ *TMS320C8x Software Toolkit*

1.2 Before You Use the TMS320C8x Simulator and Debuggers

You install the parallel debug manager, the MP debugger, the PP debugger, and the simulator core as separate files and execute them as individual tasks; however, these tasks work together to form the 'C8x simulation environment. This environment uses the interprocess communication (IPC) features of UNIX (shared memory, message queues, and semaphores) to manage communications between the different tasks that make up the simulator. If you are not sure whether the IPC features are enabled, see your system administrator.

To use the 'C8x simulation environment, you should be familiar with the IPC status (ipcs) and IPC remove (ipcrm) UNIX commands. If you use the UNIX kill (send signal) command to terminate execution of the simulator tasks, you also need to use the ipcrm command to remove the shared memory, message queues, and semaphores used by the simulator.

1.3 Step 1: Installing the Software Tools

This section explains the process of installing the software tools on your hard-disk system.

Mounting the CD-ROM

The steps to mount the CD-ROM vary according to your operating-system version:

□ If you have SunOS 4.1.x, as root, load the *TMS320C8x Software Toolkit* CD-ROM into the drive and enter the following from a command shell:

```
mount -rt hsfs /dev/sr0 /cdrom ⏎
exit ⏎
cd /cdrom ⏎
```

□ If you have SunOS 5.0 or 5.1, as root, load the *TMS320C8x Software Toolkit* CD-ROM into the drive and enter the following from a command shell:

```
mount -rF hsfs /dev/sr0 /cdrom ⏎
exit ⏎
cd /cdrom/cdrom0 ⏎
```

□ If you have SunOS 5.2 or higher:

■ If your CD-ROM drive is already attached, load the *TMS320C8x Software Toolkit* CD-ROM into the drive and enter the following from a command shell:

```
cd /cdrom/cdrom0 ⏎
```

■ If you do not have a CD-ROM drive attached, you must shut down your system to the PROM level, attach the CD-ROM drive, and enter the following:

```
boot -r ⏎
```

After you log into your system, load the CD-ROM into the drive and enter the following from a command shell:

```
cd /cdrom/cdrom0 ⏎
```

Copying the files

After you've mounted the CD-ROM, you must create the directory that will contain the software tools and copy the tools to that directory.

1) Create a directory named *mvp* on your hard disk. To create this directory, enter:

```
mkdir /pathname/mvp ⏎
```

2) Copy the files from the CD-ROM to your hard disk system:

```
cp -r * /pathname/mvp ⏎
```

Unmounting the CD-ROM

You must unmount the CD-ROM after copying the files.

□ If you have SunOS 4.1.x, SunOS 5.0, or SunOS 5.1, as root, enter the following from a command shell:

```
cd ↵
umount /cdrom ↵
eject /dev/sr0 ↵
exit ↵
```

□ If you have SunOS 5.2 or higher, enter the following from a command shell:

```
cd ↵
eject ↵
```

1.4 Step 2: Setting Up the Environment

To ensure that the tools work correctly, you must:

□ Modify the path shell variable to identify the mvp directory.

□ Define environment variables that the software tools use for finding or obtaining certain types of information.

□ Reinitialize your shell.

You can accomplish most of these tasks by entering individual commands, but it's simpler to put the commands in your shell configuration file in your home directory (for example, the .cshrc file for a C shell).

Modifying the path shell variable

You must include the software tools bin directory in your shell path. To do this, modify your .cshrc file. This file must include the pathname to your mvp/bin directory in the shell path if it is not already there. The following statement is an example of what a typical path-variable definition looks like:

```
set path = (. /bin /usr/ucb /usr/contrib/bin /usr/bin \
/usr/openwin/bin)
```

The following is an example of a modified path variable. The part of the path that is boldface is an example of a pathname that identifies the mvp/bin directory:

```
set path = (. /bin /usr/ucb /usr/contrib/bin /usr/bin \
/usr/openwin/bin /pathname/mvp/bin)
```

Setting up the environment variables

An environment variable is a special system symbol that the software tools use for finding or obtaining certain types of information. The software tools use seven environment variables: A_DIR, C_DIR, D_DIR, C_OPTION, D_OPTIONS, D_SRC, and DISPLAY. The next seven steps tell you how to set up these environment variables; these steps can be performed by modifying your .cshrc file. When defining these variables, be sure to enclose the options, filenames, or directory names within one set of quotes.

□ Define the A_DIR environment variable to identify the mvp/lib and mvp/exec/lib directories like the following:

```
setenv A_DIR "/pathname/mvp/lib;/pathname/mvp/exec/lib"
```

These directories contain the library and include files that the assembler uses.

□ Define the C_DIR environment variable to identify the mvp/lib and mvp/exec/lib directories like the following:

```
setenv C_DIR "/pathname/mvp/lib;/pathname/mvp/exec/lib"
```

These directories contain the library and include files that the compiler uses.

□ Define the D_DIR environment variable to identify the mvp/lib directory like the following:

```
setenv D_DIR "/pathname/mvp/lib"
```

This directory contains the executables and auxiliary files that you need to use the debugger and PDM.

□ You may find it useful to set the shell default options using the C_OPTION environment variable; if you do, these default options and/or input filenames are used every time you run the shell. The general format for doing this is:

setenv C_OPTION "[*compiler options*]"

Options specified with the environment variable are specified in the same way and have the same meaning as they do on the command line.

For example, if you want to always run quietly, enable symbolic debugging, and link, then set up the C_OPTION environment variable as follows.

```
setenv C_OPTION "-qg -z"
```

For more information about options, see the *C Compiler Description* chapter in the *TMS320C80 (MVP) Code Generation Tools User's Guide.*

□ You can use several options when you invoke the debugger. If you use the same options over and over, it's convenient to specify them with D_OPTIONS. The general format for doing this is:

setenv D_OPTIONS "[*object filename*] [*debugger options*]"

This tells the debugger to load the specified object file and use the specified options each time you invoke the debugger. Table 1–1 lists the options that you can identify with D_OPTIONS.

Table 1–1. Options for Use With D_OPTIONS

Option	**Description**
–b[b]	Select the screen size
–d *machinename*	Display debugger on different machine (X Windows only)
–i *pathname*	Identify additional directories
–n *processorname*	Identify the name of the processor
–o	Enable C I/O
–s	Load the symbol table only
–t *filename*	Identify a new initialization file
–v	Load without the symbol table

Note that you can override D_OPTIONS by invoking the debugger with the –x option.

For more information about options, see the *Overview of a Code Development and Debugging System* chapter in the *TMS320C80 (MVP) C Source Debugger User's Guide.*

□ Set up the D_SRC environment variable to identify any directories that contain program source files that you'll want to access from the debugger. The general format for doing this is:

setenv D_SRC "*pathname*$_1$*;pathname*$_2$*...*"

For example, if your 'C8x programs were in a directory named */user/fred/mvpsource*, the D_SRC setup would be:

```
setenv D_SRC "/user/fred/mvpsource"
```

□ If you are using the X Window System, you can use the DISPLAY environment variable to display the debugger on a different machine than the one the parallel debug manager and simulator core are running on. The general format for doing this is:

setenv DISPLAY "*machinename*"

You can also specify a different machine by using the –d debugger option (see the *Overview of a Code Development and Debugging System* chapter of the *TMS320C80 (MVP) C Source Debugger User's Guide* for more information). If you use both the DISPLAY environment variable and –d, the –d option overrides DISPLAY.

Invoking the new or modified .cshrc file

If you create or modify your .cshrc file, you must invoke that file before invoking the debugger for the first time. To do so, enter:

```
source .cshrc ⏎
```

1.5 Using the Debugger With the X Window System

When you're using the X Window System to run the 'C8x debugger, you may need to know about the keyboard's special keys, the debugger fonts, and using the debugger on a monochrome monitor.

Using the keyboard's special keys

The debugger uses some special keys that you can map differently from your particular keyboard. Some keyboards, such as the Sun Type 5 keyboard, may have these special symbols on separate keys. Other keyboards, such as the Sun Type 4 keyboard, do not have the special keys.

The special keys that the debugger uses are shown in the following table with their corresponding keysym. A **keysym** is a label that is assigned to a keystroke; it allows you to modify the action of a key on the keyboard.

Key	Keysym
F1 to F10	F1 to F10
PAGE UP	Prior
PAGE DOWN	Next
HOME	Home
END	End
INSERT	Insert
→	Right
←	Left
↑	Up
↓	Down

Use the X utility xev to check the keysyms that are associated with your keyboard. If you need to change the keysym definitions, use the xmodmap utility. For example, you could create a file that contains the following commands and use that file with xmodmap to change a Sun Type 4 keyboard to match the keys listed above:

```
keysym R13    = End
keysym Down   = Down
keysym F35    = Next
keysym Left   = Left
keysym Right  = Right
keysym F27    = Home
keysym Up     = Up
keysym F29    = Prior
keysym Insert = Insert
```

Refer to your X Window System documentation for more information about using xev and xmodmap.

Changing the debugger font

You can change the font of the debugger screen by using the xrdb utility and modifying the .Xdefaults file in your root directory. For example, to change the fonts of the MP and PP debuggers to Courier, add the following line to the .Xdefaults file:

```
mpsim*font:courier
ppsim*font:courier
```

For more information about using xrdb to change the font, refer to your X Window System documentation.

Color mappings on monochrome screens

Although a color monitor is recommended (and necessary for the graphic display features), the following table shows the color mappings for monochrome screens:

Color	Appearance on Monochrome Screen
Black	Black
Blue	Black
Green	White
Cyan	White
Red	Black
Magenta	Black
Yellow	White
White	White

Chapter 2

Installing the TMS320C8x Software Tools With Windows NT

This chapter tells you how to install release 1.13 the TMS320C8x software tools on a 32-bit x86-based or Pentium™ PC running Windows NT™ Workstation version 3.5 or later. If you are using the SunOS version of the software tools for a SPARCstation, turn to Chapter 1 for installation instructions.

The Windows NT version of the TMS320C8x software tools package is composed of the following:

- ❑ The master processor (MP) and parallel processor (PP) C compilers
- ❑ The MP and PP assemblers
- ❑ The linker
- ❑ The runtime-support and C I/O libraries

Note that the simulators and debuggers are not included in the Windows NT version of the TMS320C8x software tools. When you complete this installation, turn to Chapter 3 to verify the installation.

Chapter 4 contains documentation of tools and features that are new or have been changed since the last release. For more information about how to use the 'C8x tools, refer to the *TMS320C80 (MVP) Online Reference.*

Topic **Page**

2.1 What You'll Need

The following checklists describe items that are shipped with your 'C8x software tools and any additional items you'll need to use these tools.

Hardware checklist

□	**host**	A 32-bit x86-based or Pentium PC with an ISA/EISA bus
□	**memory**	Minimum of 16 megabytes of RAM plus 32 megabytes of hard-disk space for swap files
□	**display**	Monochrome or color (color recommended)
□	**required hardware**	A CD-ROM drive
□	**optional hardware**	A Microsoft-compatible mouse

Software checklist

□	**operating system**	Windows NT Workstation (version 3.5 or later)
□	**CD-ROMs**	*TMS320C80 (MVP) Online Reference*
□		*TMS320C8x Software Toolkit*

2.2 Step 1: Installing the Software Tools

This section explains the process of installing the software tools on a hard-disk system.

1) On your hard disk, create a directory named *c8xtools*. This directory will contain the 'C8x software tools. To create this directory, enter:

   ```
   MD C:\C8XTOOLS ⏎
   ```

2) Insert the *TMS320C8x Software Toolkit* CD-ROM into your CD-ROM drive.

3) Change to the CD-ROM drive (replace D with the name of your CD-ROM drive):

   ```
   D: ⏎
   ```

4) Copy all of the directories and files on the CD-ROM to the hard disk:

   ```
   XCOPY /S /E /V *.* C:\C8XTOOLS ⏎
   ```

   The XCOPY command copies the directories and files recursively and keeps the directory structure intact.

2.3 Step 2: Setting Up the Environment

To ensure that the tools work correctly, you must:

□ Modify the path shell variable to identify the c8xtools directory.

□ Define environment variables that the software tools use for finding or obtaining certain types of information.

You can accomplish most of these tasks by entering individual commands, but it's simpler to put the commands in your autoexec.bat file. If you modify the autoexec.bat file, be sure to invoke it before invoking the debugger for the first time. To invoke this file, enter:

```
AUTOEXEC ⏎
```

Modifying the PATH statement

To invoke the software tools without specifying the name of the directory that contains the executable files, define a path to the software tools directory. The general format is:

```
PATH=C:\C8XTOOLS
```

If you are modifying an autoexec that already contains a PATH statement, simply include **;C:\C8XTOOLS** at the end of the statement.

Setting up the environment variables

An environment variable is a special system symbol that the software tools use for finding or obtaining certain types of information. The software tools use three environment variables: A_DIR, C_DIR, and C_OPTION. The next three steps tell you how to set up these environment variables; these steps can be performed by modifying your autoexec.bat file. When defining these variables, be careful not to precede the equal sign with a space.

□ Define the A_DIR environment variable to identify the c8xtools\lib and c8xtools\exec\lib directories like the following:

```
SET A_DIR=C:\C8XTOOLS\LIB;C:\C8XTOOLS\EXEC\LIB
```

These directories contain the library and include files that the assembler uses.

□ Define the C_DIR environment variable to identify the c8xtools\lib and c8xtools\exec\lib directories like the following:

```
SET C_DIR=C:\C8XTOOLS\LIB;C:\C8XTOOLS\EXEC\LIB
```

These directories contain the library and include files that the compiler uses.

- ☐ You may find it useful to set the shell default options using the C_OPTION environment variable; if you do, these default options and/or input filenames are used every time you run the shell. The general format for doing this is:

  set C_OPTION=[*compiler options*]

  Options specified with the environment variable are specified in the same way and have the same meaning as they do on the command line.

  For example, if you want to always run quietly, enable symbolic debugging, and link, then set up the C_OPTION environment variable as follows.

  ```
  SET C_OPTION=-qg -z
  ```

  For more information about options, see the *C Compiler Description* chapter in the *TMS320C80 (MVP) Code Generation Tools User's Guide.*

Chapter 3

Verifying the Installation: A Walkthrough of the Software Tools

This chapter provides a quick walkthrough of the 'C8x software tools so that you can verify your installation and start compiling, assembling, and linking code immediately. For more information about using the code generation tools, refer to the *TMS320C80 (MVP) Code Generation Tools User's Guide.*

Topic **Page**

3.1 Assembler and Linker Walkthrough

Two tools that you will probably use often are the assembler and linker. This section helps you get started with these tools.

Step 1: Create two source files

Create the source files shown in Example 3–1 and Example 3–2; name them mp_a.asm and pp_a.s, respectively.

Example 3–1. mp_a.asm File

```
             .global    PP0_START
PP0_PARM     .set       0x010001b8
PP0_UNHALT   .set       0x30004001

MP_start:    add        -1,r0,r1                ; clears INTPEN
             wrcr       INTPEN, r1

             or         PP0_START,r0,r1
             st         PP0_PARM(r0),r1         ; PP0 start addr
             or         PP0_UNHALT, r0, r1      ; unhalt PP0
             cmnd       r1

             add        r0,r0,r1                ; clear r1
loop:        br         loop
             add        1, r1, r1               ; inc r1
```

Example 3–2. pp_a.s File

```
             .global    PP0_START

PP_DATA      .set       0x01020304
INCREMENT    .set       0x01010101
PP_ADDR      .set       0x01003400

PP0_START:   a9 = PP_ADDR
             d0 = PP_DATA
             d1 = INCREMENT
             lrse0 = 15
             nop
             nop
Loop1:       d0 = d1 + d0          ; inc data
              || *a9++ =ub d0      ; store data
             br = PP0_START
             nop
             nop
```

Step 2: Assemble the mp_a.asm file

To assemble the mp_a.asm file, enter:

mpasm mp_a ⏎

The mpasm command invokes the MP assembler. You don't have to specify the .asm file extension for the input source file (mp_a.asm), because the MP assembler uses .asm as the default extension. This example creates an object file called mp_a.obj. The assembler creates an object file if it does not encounter any errors during assembly.

Step 3: Assemble the pp_a.s file

To assemble the pp_a.s file, enter:

```
ppasm pp_a -l ⏎
```

The ppasm command invokes the PP assembler. You don't have to specify the .s file extension for the input source file (pp_a.s), because the PP assembler uses .s as the default extension. This example creates an object file called pp_a.o.

The –l (lowercase L) option tells the PP assembler to create a listing file. The listing for this example is called pp_a.lst (shown in Example 3–3). Creating a listing file is not required; however, a listing file provides you with more information and allows you to verify whether or not the assembly has resulted in the object code that you intended.

Example 3–3.pp_a.lst File

```
MVP PP Macro Assembler     Version 1.13     Wed Jun 21 16:40:27 1995
 Copyright (c) 1993-1995      Texas Instruments Incorporated
pp_a.s                                                          PAGE   1

 1                                          .global      PP0_START
 2
 3           0000000001020304 PP_DATA       .set       0x01020304
 4           0000000001010101 INCREMENT     .set       0x01010101
 5           0000000001003400 PP_ADDR       .set       0x01003400
 6
 7 00000000  9B8118C001003400 PP0_START:    a9 = PP_ADDR
 8 00000008  9B801A4001020304                d0 = PP_DATA
 9 00000010  9B811A4001010101                d1 = INCREMENT
10 00000018  9B80078000344000                lrse0 = 15
11 00000020  8800000000104100                nop
12 00000028  8800000000104100                nop
13 00000030  9A8010800050C008 Loop1:        d0 = d1 + d0          ; inc data
14                                            || *a9++ =ub d0    ; store data
15 00000038  97811BC000000000'               br = PP0_START
16 00000040  8800000000104100                nop
17 00000048  8800000000104100                nop

No Errors, No Warnings
```

Step 4: Link the mp_a.obj and pp_a.o files

To link the mp_a.obj and pp_a.o files, enter:

```
mvplnk mp_a.obj pp_a.o -m link_a.map -o prog_a.out ⏎
```

The mvplnk command invokes the 'C8x linker. The linker combines the input object files, mp_a.obj and pp_a.o, to create an executable object module called prog_a.out. The –o linker option specifies the linked module name. The –m option creates a map output file from the linking operation. Example 3–4 shows the map file resulting from this example.

Example 3–4. link_a.map File

```
*************************************************************
MVP COFF Linker                  Version 1.13
*************************************************************
Wed Jun 21 17:25:16 1995

OUTPUT FILE NAME:    <prog_a.out>
ENTRY POINT SYMBOL: 0


MEMORY CONFIGURATION

       name      origin     length    attributes       fill
       ----      ------     ------    ----------       ----
       DRAMS     00000004   000000ffc RWIX
       DRAM2     00008000   000000800 RWIX
       PRAM      01000200   000000600 RWIX
       EXTMEM    02000000   000080000 RWIX

SECTION ALLOCATION MAP

 output                                   attributes/
section   page         origin    length    input sections
-------   ----         ------    ------    --------------
.text     0            02000000  00000030
                        02000000  00000030  mp_a.obj (.text)
                        02000030  00000000  pp_a.o (.text)

.ptext    0            02000030  00000050
                        02000030  00000050  pp_a.o (.ptext)

.bss      0            02000000  00000000  UNINITIALIZED
                        02000000  00000000  mp_a.obj (.bss)
                        02000000  00000000  pp_a.o (.bss)

.data     0            02000000  00000000  UNINITIALIZED
                        02000000  00000000  mp_a.obj (.data)
                        02000000  00000000  pp_a.o (.data)

.pbss     0            00000004  00000000  PASS SECTION
```

Example 3–4. link_a.map File (Continued)

```
GLOBAL SYMBOLS

address   name                                   address   name
-------   ----                                   -------   ----
02000000  .bss                                   02000000  end
02000000  .data                                  02000000  .data
02000000  .text                                  02000000  edata
02000030  PP0_START                              02000000  .bss
02000000  edata                                  02000000  .text
02000000  end                                    02000030  PP0_START
02000030  etext                                  02000030  etext

[7 symbols]
```

3.2 C Compiler Walkthrough

The TMS320C8x C compilers operate in two passes.

- ❑ The first pass parses the code.
- ❑ The second pass produces a single assembly language source file that must be assembled and linked.

The simplest way to compile, assemble, and link a C program is to use the shell program, which is included with the compilers. This section helps you get started with compiling C programs.

Step 1: Create a sample C file

Create a sample file called function.c that contains the code shown in Example 3–5.

Example 3–5. function.c File

```
/*************************************/
/*           function.c              */
/*  (sample file for walkthrough)    */
/*************************************/
main ()
{
    int x = -3;
    x = abs_func(x);
}

int abs_func(int i)
{
    int temp = i;
    if (temp < 0) temp *= -1;
    return (temp);
}
```

Step 2: Assemble and compile function.c

To invoke the shell program to compile and assemble function.c, enter:

`mpcl function.c` ↵

As the shell program compiles, it displays the information shown in Example 3–6.

Example 3–6. Shell Program Messages

```
[function.c]
MVP MP ANSI C Compiler     Version 1.13
Copyright (c) 1993-1995    Texas Instruments Incorporated
   "function.c" ==> main
   "function.c" ==> abs_func
MVP MP ANSI C Codegen      Version 1.13
Copyright (c) 1993-1995    Texas Instruments Incorporated
   "function.c": ==> main
   "function.c": ==> abs_func
MVP MP Macro Assembler     Version 1.13
Copyright (c) 1993-1995    Texas Instruments Incorporated
 PASS 1
 PASS 2

 No Errors,  No Warnings
```

The shell program runs two compiler passes and the assembler as follows:

□ mpac → MP C parser
□ mpcg → MP code generator
□ mpasm → MP assembler

Inspecting the assembly language output

By default, the shell program deletes the assembly language file from the compiler after it is assembled. If you want to inspect the assembly language output, use the –k option to retain the assembly language file:

`mpcl function.c -k` ↵

Changing the output file

Also by default, the shell program creates a COFF object file as output; however, if you use the –z option, the output is an executable COFF object module. The following examples show the two ways of creating an executable object module:

- To create an executable object module, link the object file with the runtime-support library mp_rts.lib:

  ```
  mvplnk -c function -o function.out -l mp_rts.lib ⏎
  ```

  This example uses the –c linker option because the code came from a C program. The –l option tells the linker that the input file mp_rts.lib is an object library. The –o option names the output module function.out. If you don't use the –o option, the linker uses a.out as the default name.

- In this example, use the –z option, which tells the shell program to run the linker. All other linker options should follow –z.

  ```
  mpcl function.c -z -o function.out -l mp_rts.lib ⏎
  ```

  This example runs the two compiler passes, the assembler, and the linker as follows:

  - mpac → MP C parser
  - mpcg → MP code generator
  - mpasm → MP assembler
  - mvplnk → 'C8x linker

Using the interlist utility

The TMS320C8x compiler package also includes an interlist utility. This program interlists the C source statements as comments in the assembly language compiler output, allowing you to inspect the assembly language generated for each line of C. To run the interlist utility, invoke the shell program with the –s option:

```
mpcl function.c -s -z -o function.out -l mp_rts.lib ⏎
```

The output of the interlist utility is written to the assembly language file created by the compiler. (The –s shell option implies the –k option; that is, when you use the interlist utility, the assembly file is automatically retained.)

Chapter 4

Release Notes

This chapter contains documentation of tools and features that are new or have been changed since the last release. It is not an exhaustive list of all code changes since the last release, but it is a list of all of the changes that may require modifications to your C source, assembly source, linker control files, debugger batch or initialization files, or makefiles, so please take time to read this section completely.

Topic **Page**

4.1 Media Contents

The TMS320C8x software tools are supported on SPARCstations with SunOS and on PCs with Windows NT. Table 4–1 through Table 4–4 list the files shipped on the *TMS320C8x Software Toolkit* CD-ROM.

Table 4–1. Code Generation Tools

Windows NT File	SunOS File	Description		
clist.exe	clist	C source interlist utility		
mpac.exe	mpac	MP ANSI C parser		
mpasm.exe	mpasm	MP assembler		
mpcg.exe	mpcg	MP ANSI C code generator		
mpcl.exe	mpcl	MP compiler shell program		
mpmk.exe	mpmk	MP library build utility		
mpopt.exe	mpopt	MP optimizer		
mvpar.exe	mvpar	’C8x archiver		
mvphex.exe	mvphex	’C8x hex conversion utility		
mvplnk.exe	mvplnk	’C8x COFF linker		
ppac.exe	ppac	PP ANSI C parser		
ppasm.exe	ppasm	PP assembler		
ppcg.exe	ppcg	PP assembler		
ppcl.exe	ppcl	PP compiler shell program		
ppmk.exe	ppmk	PP library build utility		
ppopt.exe	ppopt	PP optimizer		
*.h	*.h	#include header files for RTS:		
		assert.h	ctype.h	errno.h
		float.h	limits.h	math.h
		mvp.h	setjmp.h	stdarg.h
		stddef.h	stdio.h	stdlib.h
		string.h	time.h	

Table 4–2. Libraries

Windows NT File	SunOS File	Description
mp_cio.lib	mp_cio.lib	MP ANSI-standard I/O—big endian
mp_ciol.lib	mp_ciol.lib	MP ANSI-standard I/O—little endian
pp_cio.lib	pp_cio.lib	PP ANSI-standard I/O—big endian
pp_ciol.lib	pp_ciol.lib	PP ANSI-standard I/O—little endian
mp_rts.lib	mp_rts.lib	MP runtime-support functions—big endian
mp_rtsl.lib	mp_rtsl.lib	MP runtime-support functions—little endian
pp_rts.lib	pp_rts.lib	PP runtime-support functions—big endian
pp_rtsl.lib	pp_rtsl.lib	PP runtime-support functions—little endian
mp_rts.src	mp_rts.src	Source library for mp_rst.lib and mp_rstl.lib
pp_rts.src	pp_rts.src	Source library for pp_rst.lib and pp_rstl.lib

Table 4–3. Sample Link Control Files

Windows NT File	SunOS File	Description
mpcio.cmd	mpcio.cmd	MP ANSI-standard I/O—big endian
mpciol.cmd	mpciol.cmd	MP ANSI-standard I/O—little endian
ppcio.cmd	ppcio.cmd	PP ANSI-standard I/O—big endian
ppciol.cmd	ppciol.cmd	PP ANSI-standard I/O—little endian
mplnk.cmd	mplnk.cmd	MP runtime-support functions—big endian
mplnkl.cmd	mplnkl.cmd	MP runtime-support functions—little endian
pplnk.cmd	pplnk.cmd	PP runtime-support functions—big endian
pplnkl.cmd	pplnkl.cmd	PP runtime-support functions—little endian

Table 4–4. Simulator and Debugger Files (SunOS Only)

SunOS File	Description
pdm	Parallel debug manager environment
mpsim	MP C source debugger and simulator core
ppsim	PP C source debugger and simulator core
simMVP	'C8x simulator core (the mpsim and ppsim files invoke simMVP)
init.cmd	A general-purpose batch file that contains debugger commands. This batch file, shipped with the debugger, defines an MP and PP memory map. If this file isn't present when you invoke the debugger, then all memory is invalid at first. When you first start using the debugger, this memory map should be sufficient for your needs. Later, you may want to define your own memory map. For information about setting up your own memory map, refer to the Defining a Memory Map chapter in the *TMS320C80 (MVP) C Source Debugger User's Guide.*
init.pdm	A general-purpose batch file that contains special parallel debug manager commands. These commands allow you to group and send commands to debuggers under the control of the parallel debug manager. The parallel debug manager reads this file during invocation.
init.clr	A general-purpose screen configuration file. If init.clr isn't present when you invoke the debugger, the debugger uses the default screen configuration. For information about this file and about setting up your own screen configuration, refer to the Customizing the Debugger Display chapter in the *TMS320C80 (MVP) C Source Debugger User's Guide.*

4.2 Changes to the Debugger

CACHEVIEW command

Previous versions of the MP debugger accessed memory from the processor's reference through the data caches. This feature restricted access to external memory when data was found within the MP's data cache.

The CACHEVIEW command was added to this version. This command toggles the view of all memory accesses generated by the debugger between the data cache view and the external memory view. When the data cache view is enabled (default), memory writes are performed to both the cache and the external memory location (cache write through). When the external memory view is enabled, memory writes are performed only with the external memory location.

C I/O enable option (–o option)

If you plan to debug a program that uses the standard C I/O facilities, you must invoke the MP or PP debugger with the –o option. When you use the –o option, the debugger sets up a special environment with the host machine for performing I/O functions. This environment will not be set up if you don't use the –o option.

The –o option was added to prevent more than one debugger from attempting to set up the special C I/O environment when multiple 'C8x debuggers are running. Since only one debugger (or processor) can be using the C I/O facilities at a time, the –o option allows you to explicitly identify which debugger will set up the environment.

This change may require you to modify the PDM initialization files or script files that you use to invoke the debuggers.

4.3 Changes That Affect the Assemblers and the Compilers

Support for production silicon version 3

You can use the –v3 option with the PP compiler (ppcl) and PP assembler (ppasm) commands to enable new features for the PP in production silicon version 3. To disable these features, you can use the –v2 ppcl and ppasm options (default).

For more information about the –v3 and –v2 options, see the *C Compiler Description* chapter in the *TMS320C80 (MVP) Code Generation Tools User's Guide.*

Little-endian support

You can use the –me option with the PP compiler (ppcl), MP compiler (mpcl), MP assembler (mpasm), and PP assembler (ppasm) commands to indicate that little-endian code should be generated.

Four new libraries, pp_rtsl.lib, mp_rtsl.lib, pp_ciol.lib, and mp_ciol.lib, have been added to the toolset. These libraries contain the little-endian object code for the runtime-support and C I/O routines for the PP and MP.

For more information about the –me option, see the *C Compiler Description* chapter in the *TMS320C80 (MVP) Code Generation Tools User's Guide.*

Note:

The simulator does not support little-endian code.

PP default filename extensions

The PP compiler and assembler (ppcl and ppasm) now use different default filename extensions for assembly and object code. The old default extensions were .asm for assembly files and .obj for object files. The new default extensions are .s for assembly files and .o for object files.

The MP compiler and assembler (mpcl and mpasm) continue to use the .asm and .obj extensions. The PP extensions were changed to make it easier for you to write default rules for makefiles. For example, .c.o implies the PP C compiler; .c.obj implies the MP C compiler.

This change may require you to modify any makefiles or linker command files that you use to build PP code.

For more information about filename extensions, see the *C Compiler Description* chapter in the *TMS320C80 (MVP) Code Generation Tools User's Guide.*

Nondata registers legal to barrel shift on the PP

The restriction that only data registers may be barrel shifted in the PP before input to the B port of the ALU has been removed. The PP assembler now accepts all registers as input to the barrel shifter. The PP compiler now generates instructions that shift nondata registers.

4.4 Changes to the Compilers

MP cache bypass reference mechanism (old use of the volatile keyword)

In previous revisions of the MP compiler, the volatile keyword was used to indicate that a reference should bypass cache. This use of the volatile keyword is no longer supported. However, you can use the –mv option with mpcl to retain the old semantics of volatile. The –mv option should ease the transition to the version 1.13 tools if you have existing code that uses the volatile keyword as a means for bypassing the cache. However, volatile is not suggested as a mechanism for bypassing cache, because there are some inherent problems with its use, and it does not allow a variable to be both volatile and accessed normally.

The version 1.13 MP compiler uses macros to indicate a cache bypass reference. The following macros have been provided in mvp.h:

- ❑ NOCACHE_CHAR(x)
- ❑ NOCACHE_UCHAR(x)
- ❑ NOCACHE_SHORT(x)
- ❑ NOCACHE_USHORT(x)
- ❑ NOCACHE_INT(x)
- ❑ NOCACHE_UINT(x)
- ❑ NOCACHE_LONG(x)
- ❑ NOCACHE_ULONG(x)
- ❑ NOCACHE_FLOAT(x)
- ❑ NOCACHE_DOUBLE(x)
- ❑ NOCACHE_LDOUBLE(x)

These macros can be used on either side of an assignment, as shown here:

```
int x,y,z;

x = NOCACHE_INT(y);     ; cache bypass load of y
NOCACHE_INT(z) = x;     ; cache bypass store of z
```

Each macro is used to reference memory as the type indicated in the name of the macro; that is, x = NOCACHE_INT(y) loads a signed 32-bit value at an address represented by y, and x = NOCACHE_USHORT(y) loads an unsigned 16-bit value at an address represented by y.

The macros take the address of their argument; the argument *must* be a valid lvalue (that is, something that is legal to assign to).

For more information about bypassing cache and the nocache macros, see the *MP Runtime Environment* chapter in the *TMS320C80 (MVP) Code Generation Tools User's Guide.*

PP C variable prefix

The 'C8x C compilers attach a prefix to all variables. This prefix is used to avoid name conflicts with assembly language code labels. In previous releases, the prefix for both the MP and PP compilers was an underscore (_). In this release, the prefix for PP C compiled symbol names has changed from an underscore to a dollar sign ($). This change helps avoid name conflicts with MP C compiled symbols, whose prefix remains an underscore.

The modification requires you to change the name of any symbol defined in assembly language that was intended to be referenced from PP C code. You simply need to change the underscore prefix to a dollar-sign prefix. If the symbol is to be referenced from MP and PP C code, both prefixed names must be defined. For example:

```
_label:
$label:      nop
```

The modification also requires you to change any MP C code that references a PP C symbol, since the names now differ. The new shared keyword or SHARED pragma needs to be used. The shared keyword and SHARED pragma are described on page 4-11.

Note:

When specifying a PP C symbol name on the command line using ppcl or mvplnk, the UNIX shell tries to interpret the dollar sign character, so you must use an escape sequence in on the command line.

For example, in the command:

```
mvplnk file.obj -u \$exit -l pp_cio.lib -l -o file.out
```

the $ in $exit is escaped with the backslash (\) to prevent $exit from being interpreted as a UNIX shell variable.

PP C compiler section names

Some of the section names that the PP C compiler uses by default have been changed to avoid allocation conflicts with similarly named MP C section names. The following list indicates the changes that were made.

Old PP Section	New PP Section
.bss	.pbss
.ext	.bss
.cinit	.pcinit
.stack	.pstack
.sysmem	.psysmem

This change requires modification to any linker control file that you use to link PP C programs. The sample linker command files (pplnk.cmd and mplnk.cmd) included in this release illustrate the modifications.

For more information about the sections used by the PP compiler, see the *PP Runtime Environment* chapter in the *TMS320C80 (MVP) Code Generation Tools User's Guide*.

PP C autoinitialization

The mechanics of the PP C compiler's autoinitialization have changed in the following ways:

- ❑ The PP C init table is now in the .pcinit section instead of .cinit.
- ❑ The $pcinit label (instead of cinit) refers to the start of the PP C init table.
- ❑ The PP C init table is now aligned to a 4-byte boundary instead of an 8-byte boundary.
- ❑ The –pc linker option is now used instead of –c to indicate ROM model initialization for PP C code. The –pc option pads the .pcinit section with the termination record and assigns a value to the symbol $pcinit to indicate the start of the PP C init tables.

Autoinitialization on the MP remains the same.

The mpcl shell automatically issues the –c linker option if the –z option is specified. The ppcl shell automatically issues the –pc linker option if the –z option is specified.

For more information about the –pc linker option, see the *Linker Description* chapter in the *TMS320C80 (MVP) Code Generation Tools User's Guide*. For more information about autoinitialization on the PP, see the *C Compiler Description* and *PP Runtime Environment* chapters in the *TMS320C80 (MVP) Code Generation Tools User's Guide*.

MP and PP C shared keyword and SHARED pragma

The SHARED pragma and the shared keyword have been added to both the MP and PP compilers to allow C symbol names to be seen both in MP and PP C source code. The syntax for the SHARED pragma is:

#pragma SHARED(*var***)**
int *var***;**

The syntax for the shared keyword is:

shared int *var***;**

The first effect of using the shared keyword or SHARED pragma is that the compiler generates variable definitions with two labels: one with an underscore prefix and one with a dollar sign prefix. This allows the symbol name to be referenced from both MP C and PP C.

The other result of the shared keyword or SHARED pragma is that neither the underscore prefixed variable nor the dollar-sign prefixed variable is hidden by a task-level link (–t linker option) or when linking with the –h (hide symbols) option. Task-level linking is described on page 4-17. Shared symbols are not changed to static in a task-level link, because a .system directive is generated for the symbols instead of a .global directive.

The addition of the shared keyword requires you to change any symbol name that you have called **shared** (in all lower case) in either MP or PP C code.

For more information about the shared keyword and the SHARED pragma, see the *TMS320C8x C Language* chapter in the *TMS320C80 (MVP) Code Generation Tools User's Guide*.

New PP C sharedpp keyword and SHAREDPP pragma

The SHAREDPP pragma and the sharedpp keyword have been added to the PP compiler only. SHAREDPP informs the compiler to generate a .system directive for the symbol instead of a .global directive. This results in the symbol remaining global even after a task-level link. The difference between SHAREDPP and SHARED is that for SHAREDPP, both underscore and dollar-sign prefixed labels are not generated; only the dollar-sign prefixed symbol is created. Therefore, the symbol is not visible to MP code.

This keyword/pragma is useful for PP functions that can be shared among all PP tasks. The function needs to be linked in only once, not in each task-level link.

The addition of the sharedpp keyword requires you to change any symbol name that you have called sharedpp (in all lower case) in PP C code.

For more information about the sharedpp keyword and the SHAREDPP pragma, see the *TMS320C8x C Language* chapter in the *TMS320C80 (MVP) Code Generation Tools User's Guide*.

4.5 Changes to the Assemblers

MP and PP assembler directive (.system)

The .system assembler directive has been added to both the MP and PP assemblers. The .system directive is identical to the .global assembler directive, except for how symbols declared with the .system directive are handled by the linker during a task-level link (–t linker option) or when linking with the –h (hide symbols) option. Task-level linking is described on page 4-17. The –h linker option is described in the *Linker Description* chapter in the *TMS320C80 (MVP) Code Generation Tools User's Guide.*

When task-level linking (–t) or when linking with the –h option, the linker changes symbols declared with the .global assembler directive to static symbols, thus hiding their visibility to any further link steps. The linker does not change symbols declared with the .system directive to static; they remain external.

The addition of the .system directive does not require any modifications to existing code, makefiles, or linker control files.

For more information about the .system directive, see the *Assembler Directives* chapter in the *TMS320C80 (MVP) Code Generation Tools User's Guide.*

Predefined macro function names

The names of the following predefined assembler macro functions have been changed. The new names are the same as the old names, with an additional dollar sign prepended. This change is needed to avoid conflicts with PP C variable names. This change is in both the MP and PP assemblers.

Old Name	New Name
$symlen	$$symlen
$symcmp	$$symcmp
$firstch	$$firstch
$lastch	$$lastch
$isdefed	$$isdefed
$ismember	$$ismember
$iscons	$$iscons
$isname	$$isname
$isreg	$$isreg

For more information about the these macros, see the *Macro Language* chapter in the *TMS320C80 (MVP) Code Generation Tools User's Guide.*

4.6 Changes to the Linker

PP stack and heap size linker options

Two new linker options were added to control the size of the PP stack and heap sections:

- ❑ The –pstack option controls the size of the PP stack by changing the size of the .pstack section. The linker sets the $_STACK_SIZE symbol to reflect the size of the .pstack section. This option is now used for PP C code links instead of –stack.
- ❑ The –pheap option controls the size of the PP heap by changing the size of the .psysmem section. The linker sets the $_SYSMEM_SIZE symbol to reflect the size of the .psysmem section. This option is now used for PP C code links instead of –heap.

The MP stack and heap are still controlled by the –stack and –heap linker options.

The default sizes for the MP and PP stack and heap have also changed:

Section	Old Default	New Default
MP stack	128 bytes	1024 bytes
MP heap	128 bytes	1024 bytes
PP stack	128 bytes	128 bytes
PP heap	64 bytes	128 bytes

These changes require you to modify any makefiles or linker command files that you use to set up the PP stack or heap for PP C code.

For more information about controlling the MP and PP stack and heap sections, see the *Linker Description* chapter in the *TMS320C80 (MVP) Code Generation Tools User's Guide.*

PASS section attribute in linker command files

The PASS section capability has been added so that a section can be allocated space in the memory map in a link step and then be ignored (or passed through) in any further link steps.

This attribute is specified by using the PASS keyword in a linker command file. For example:

```
.pbss : (PASS) > DRAM0
```

This example informs the linker that it should group all input sections with the name .pbss and create one output section called .pbss. The output section .pbss has the attribute PASS attached to it.

The effect of PASS happens when the resulting output file is used later as input for further linking. When the linker sees an input section that has the PASS attribute, it:

- ❑ Does not group it with any other section
- ❑ Does not change the address that was assigned to the section
- ❑ Does not prevent other sections from being allocated into the same address space

In other words, it is ignored and is simply passed through. This is very useful on a 'C8x device for doing task-level linking of PP code and allowing the uninitialized sections to overlay each other. Task-level linking is described on page 4-17.

The addition of the PASS attribute does not require any modifications of source, makefiles, or linker control files.

For more information about the PASS attribute, see the *Linker Description* chapter in the *TMS320C80 (MVP) Code Generation Tools User's Guide.*

Supporting task-level links (–t option)

Task-level linking allows you to have symbols that are global within a subset of the files and hidden from the rest of the files. This is useful for a 'C8x device when linking together several PP applications.

The –t *name* linker option causes the following things to occur:

□ All symbols declared as .global are changed to static, thus hiding them from further link steps.

□ All symbols declared as .system remain .system and are global (that is, they are visible to further links).

□ Two symbols, $ep_name and _ep_name (where name is the argument you supply to the –t option), are created and equated to the entry point symbol. This is useful for PP C code, since the entry point in the boot routine is always _c_int00. This provides a way to distinguish between different task entry points. The two symbols are not created if an entry point symbol is not specified.

□ A dummy filename entry is created in the symbol table with the name name.c, where name is the name supplied to the –t option. All of the (non-function name) task-level globals (that is, all of the former .global symbols that were converted to statics) are associated with this dummy filename. This allows you to reference them in the debugger using the filename.variable mechanism. For example, if you declare a variable xyz as .global and you task-level link with –t mytask, then in the debugger, you can refer to the variable xyz as mytask.xyz. Function names that are changed to static by the –t linker option are associated with the file in which they were defined.

Note:

The task name you supply to the –t option should not be a name that you have used in the code. It should be unique. If it is the same as another symbol, then the filename.variable mechanism will not work in the debugger.

The addition of the –t linker option does not require any modifications to existing source code, makefiles, or linker control files.

For more information about task-level linking with the –t linker option, see the *Linker Description* chapter in the *TMS320C80 (MVP) Code Generation Tools User's Guide.*

Default linker MEMORY and SECTIONS directives for a ’C8x device

The default linker MEMORY and SECTIONS directives have changed, as illustrated here:

```
MEMORY
{
    DRAMS    : origin = 0x0004    , length = 0x0ffc
    DRAM2    : origin = 0x8000    , length = 0x0800
    PRAM     : origin = 0x01000200, length = 0x0600
    EXTMEM   : origin = 0x02000000, length = 0x80000
}

SECTIONS
{
    .text    : ALIGN(16) {} > EXTMEM
    .ptext   : ALIGN(16) {} > EXTMEM
    .bss     : ALIGN(4 ) {} > EXTMEM
    .data    : ALIGN(4 ) {} > EXTMEM
    .const   : ALIGN(4 ) {} > EXTMEM                ;cflag option only
    .switch  : ALIGN(4 ) {} > EXTMEM                ;cflag option only
    .sysmem  : ALIGN(4 ) {} > EXTMEM                ;cflag option only
    .stack   : ALIGN(4 ) {} > EXTMEM                ;cflag option only
    .cinit   : ALIGN(4 ) {} > EXTMEM                ;cflag option only
    .pcinit  : ALIGN(4 ) {} > EXTMEM                ;pcflag option only
    .pbss    : ALIGN(4 ) {} (PASS) > DRAMS
    .psysmem : ALIGN(4 ) {} (PASS) > DRAM2          ;pcflag option only
    .pstack  : ALIGN(4 ) {} (PASS) > PRAM           ;pcflag option only
}
```

For more information about default linker memory allocation, see the *Linker Description* chapter in the *TMS320C80 (MVP) Code Generation Tools User’s Guide.*

Handling symbols (–g option)

The –g *name* linker option informs the linker that the symbol *name* should not be changed to static during a task-level link (–t) or a link using the –h (hide symbols) option. This option allows you to effectively handle a symbol in the linker as though it had been declared with the .system assembler directive. Task-level linking is described on page 4-17.

For more information about the –g and –h linker options, see the *Linker Description* chapter in the *TMS320C80 (MVP) Code Generation Tools User’s Guide.*

Index

NOTES

TI Worldwide Sales and Representative Offices

AUSTRALIA / NEW ZEALAND: Texas Instruments Australia **Ltd.: Melbourne** [61] 3-696-1211, Fax 3-696-4446; Sydney 2-910-3100, Fax 2-805-1186.

BELGIUM: Texas Instruments Belgium S.A./N.V.: Brussels [32] (02) 726 75 80, Fax (02) 726 72 76.

BRAZIL: Texas Instrumentos Electronicos do Brasil **Ltda.: Sao Paulo** [55] 11-535-5133.

CANADA: Texas Instruments Canada Ltd.: Montreal (514) 421-2750; **Ottawa** (613) 726-3201; Fax 726-6363; **Toronto** (905) 884-9181; Fax 884-0062.

DENMARK: Texas Instruments A/S: Ballerup [45] (44) 68 74 00.

FRANCE/MIDDLE EAST/AFRICA: Texas Instruments **France: Velizy-Villacoublay** [33] (1) 30 70 10 01, Fax (1) 30 70 10 54.

GERMANY: Texas Instruments Deutschland GmbH.: Freising [49] (08161) 800, Fax (08161) 80 45 16; **Hannover** (0511) 90 49 60, Fax (0511) 64 90 331; **Ostfildern** (0711) 340 30, Fax (0711) 340 32 57.

HONG KONG: Texas Instruments Hong Kong **Ltd.:** Kowloon [852] 2956-7288, Fax 2956-2200.

HUNGARY: Texas Instruments Representation: Budapest [36] (1) 269 83 10, Fax (1) 267 13 57.

IRELAND: Texas Instruments Ireland Ltd.: Dublin [353] (01) 475 52 33, Fax (01) 478 14 63.

ITALY: Texas Instruments Italia S.p.A.: Agrate Brianza [39] (039) 684 21, Fax (039) 684 29 12; **Rome** (06) 657 26 51.

JAPAN: Texas Instruments Japan Ltd.: Kanazawa [81] 0762-23-5471, Fax 0762-23-1583; **Kita Kanto** 0485-22-2440, Fax 0485-23-5787; **Kyoto** 075-341-7713, Fax 075-341-7724; **Kyushu** 0977-73-1557, Fax 0977-73-1583; **Matsumoto** 0263-33-1060, Fax 0263-35-1025; Nagoya 052-232-5601, Fax 052-232-7888; **Osaka** 06-204-1881, Fax 06-204-1895; Tachikawa 0425-27-6760, Fax 0425-27-6426; **Tokyo** 03-3769-8700, Fax 03-3457-6777; **Yokohama** 045-338-1220, Fax 045-338-1255.

KOREA: Texas Instruments Korea Ltd.: Seoul [82] 2-551-2804, Fax 2-551-2828.

MAINLAND CHINA: Texas Instruments China Inc.: Beijing [86] 10-500-2255, Ext. 3750, Fax 10-500-2705.

MALAYSIA: Texas Instruments Malaysia Sdn Bhd: Kuala Lumpur [60] 3-208-6001, Fax 3-230-6605.

MEXICO: Texas Instruments de Mexico S.A. de C.V.: **Colonia del Valle** [52] 5-639-9740.

NORWAY: Texas Instruments Norge A/S: Oslo [47] (02) 264 75 70.

PHILIPPINES: Texas Instruments Asia Ltd.: Metro Manila [63] 2-636-0980, Fax 2-631-7702.

SINGAPORE (& INDIA, INDONESIA, THAILAND): Texas Instruments Singapore (PTE) Ltd.: Singapore [65] 390-7100, Fax 390-7062.

SPAIN/PORTUGAL: Texas Instruments España S.A.: Madrid [34] (1) 372 80 51, Fax (1) 307 68 64.

SUOMI/FINLAND: Texas Instruments/OY: Espoo [358] (0) 43 54 20 33, Fax (0) 46 73 23.

SWEDEN: Texas Instruments International Trade Corporation (Sverigefilialen): Kista [46] (08) 752 58 00, Fax (08) 751 97 15.

SWITZERLAND: Texas Instruments Switzerland AG: Dietikon [41] 886-2-3771450.

TAIWAN: Texas Instruments Taiwan Limited: Taipei [886] 2-378-6800, Fax 2-377-2718.

THE NETHERLANDS: Texas Instruments Holland, B.V. Amsterdam [31] (020) 546 98 00, Fax (020) 646 31 36.

UNITED KINGDOM: Texas Instruments Ltd.: Northampton [44] (01604) 66 30 00, Fax (01604) 66 30 01.

TEXAS INSTRUMENTS

UNITED STATES: Texas Instruments Incorporated: ALABAMA: Huntsville (205) 430-0114; **ARIZONA: Phoenix** (602) 224-7800; **CALIFORNIA: Irvine** (714) 660-1200; **Los Angeles** (818) 704-8100; **San Diego** (619) 278-9600; **San Jose** (408) 894-9000; **COLORADO: Denver** (303) 488-9300; **CONNECTICUT: Wallingford** (203) 269-0074; **FLORIDA: Fort Lauderdale** (305) 425-7805; **Orlando** (407) 667-5308; **Tampa** (813) 573-0331; **GEORGIA: Atlanta** (770) 662-7967; **ILLINOIS: Chicago** (708) 517-4500; **INDIANA: Indianapolis** (317) 573-6400; **KANSAS: Kansas City** (913) 451-4511; **MARYLAND: Baltimore** (410) 312-7900; **MASSACHUSETTS: Boston** (617) 895-9100; **MICHIGAN: Detroit** (810) 305-5700; **MINNESOTA: Minneapolis** (612) 828-9300; **NEW JERSEY: Edison** (908) 906-0033; **NEW MEXICO: Albuquerque** (505) 345-2555; **NEW YORK: Long Island** (516) 454-6601; **Poughkeepsie** (914) 897-2900; **Rochester** (716) 385-6770; **NORTH CAROLINA: Charlotte** (704) 522-5487; **Raleigh** (919) 876-2725; **OHIO: Cleveland** (216) 328-2149; **Dayton** (513) 427-6200; **OREGON: Portland** (503) 643-6758; **PENNSYLVANIA: Philadelphia** (610) 825-9500; **PUERTO RICO: Hato Rey** (809) 753-8700; **TEXAS: Austin** (512) 250-6769; **Dallas** (214) 917-1264; **Houston** (713) 778-6592; **Midland** (915)561-6521; **WISCONSIN: Milwaukee** (414) 798-5021.

North American Authorized Distributors

COMMERCIAL	
Almac / Arrow	800-426-1410 / 800-452-9185 Oregon only
Anthem Electronics	800-826-8436
Arrow / Schweber	800-777-2776
Future Electronics (Canada)	800-388-8731
Hamilton Hallmark	800-332-8638
Marshall Industries	800-522-0084 or www.marshall.com
Wyle	800-414-4144
OBSOLETE PRODUCTS	
Rochester Electronics	508-462-9332
MILITARY	
Alliance Electronics Inc	800-608-9494
Future Electronics (Canada)	800-388-8731
Hamilton Hallmark	800-332-8638
Zeus, An Arrow Company	800-524-4735
CATALOG	
Allied Electronics	800-433-5700
Arrow Advantage	800-777-2776
Newark Electonics	800-367-3573

For Distributors outside North America, contact your local Sales Office.

Important Notice: Texas Instruments (TI) reserves the right to make changes to or to discontinue any product or service identified in this publication without notice. TI advises its customers to obtain the latest version of the relevant information to verify, before placing orders, that the information being relied upon is current.

Please be advised that TI warrants its semiconductor products and related software to the specifications applicable at the time of sale in accordance with TI's standard warranty. TI assumes no liability for applications assistance, software performance, or third-party product information, or for infringement of patents or services described in this publication. TI assumes no responsibility for customers' applications or product designs.

A111395

Printed in the USA

TEXAS INSTRUMENTS

Printed in U.S.A., June 1995
2613611-9741 revision C